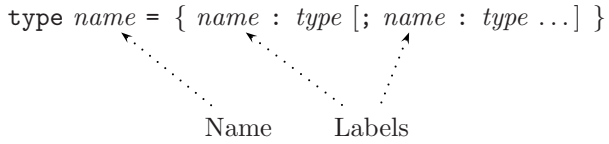
1 Objectives

Even more powerful than the idea of recursion is the idea of abstraction—which is a lot easier to implement if you can make your own types. Your goal after this lecture is to...

- Be able to describe the record type and the disjoint type.
- Be able to give an example of how each one is used.
- Be able to draw a memory diagram that is the result of allocating a variable of a disjoint type.
- Be able to use the `match/with` syntax to deconstruct a disjoint type.
- Be able to describe how data-structures in a functional language "recycle" data from previous versions of the structure when a modification has been made.

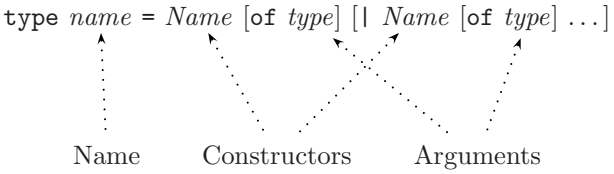
2 Examples

2.1 Record Types




```
# type complex = { re : float; im : float };;
type complex = { re : float; im : float; }
# let cadd x y = { re = x.re +. y.re;
                   im = x.im +. y.im };;
val cadd : complex -> complex -> complex = <fun>
```

2.2 Disjoint Types



- Note: Constructor names must be capitalized.
- Constructor names also must be unique.

```
# type contest = Rock | Scissors | Paper
# type velocity = MeterPerSecond of float
                | FeetPerSecond of float;;
```

2.3 Option Type

```
# type 'a option = Some of 'a | None;;
# let rec getItem key lst =
    match lst with
    | [] -> None    note the type variables!
    | (k,v)::xs -> if key = k then Some v
                   else getItem key xs;;
# getItem 3 [2,"french hens"; 3,"turtle doves"];;
- : string option = Some "turtle doves"
# getItem 5 [2,"french hens"; 3,"turtle doves"];;
- : string option = None
```

3 Problems

Here are some example problems which will be useful to study in preparation for the exam. Some of these may be done in class.

1. Write a function to multiply complex numbers. Use the type definition we did in class.
2. Write the type defintion for a phone book entry? Assume we want to store a name, address, and phone number, all as strings.
3. Write the type for a binary search tree. In this version, let a "node" be something that has data, and a "leaf" does not have any data. Use parametric polymorphism.
4. For the BST type above, write `find` and `add`.
5. Write a disjoint type called `'a tsil`, which is a singly linked list, only the link comes first, and the data comes second. Have constructors `Snoc` and `Lin`.
6. For the `tsil` type above, write a function to convert back and forth with OCaml lists.
7. What does it mean when we say that a data-structure is handled using functional style? What are some advantages and disadvantages of this style?
8. Write the `'a option` type definition.
9. Write a function that adds two variables of type `int option`.
10. What is the use of the `'a option` type?